E.L.E.N. : Un Système d'interrogation d'une base de logiciels

Jean-Pierre Chevallet

▶ **To cite this version:**

Jean-Pierre Chevallet. E.L.E.N. : Un Système d'interrogation d'une base de logiciels. Congrès INFORSID 1991, 1991, Paris, France. pp.1–20. hal-00954073

HAL Id: hal-00954073
https://inria.hal.science/hal-00954073

Submitted on 3 Mar 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INFORSID 1991

E.L.E.N. : UN SYSTèME D'INTERROGATION D'UNE BASE DE LOGICIELS

CHEVALLET Jean-Pierre

LGI-IMAG BP 53 X
38041 Grenoble Cedex France
Tel : 76 51 46 00 ext 5113
E. Mail : chevalet@imag.imag.fr

Résumé :

L'augmentation de la taille et de la durée de vie des logiciels fait ressortir certains problèmes liés à l'activité du génie logiciel parmi lesquels : la cohérence entre les documents et les logiciels, la gestion des versions, le morcellement de l'information parmi les participants au projet, la vérification et la correction du logiciel, la recherche de codes réutilisables, etc. Une gestion complète et uniforme des logiciels et de leurs documentations, ainsi que la possibilité d'interroger les informations ainsi gérées, permet de simplifier un certain nombre de ces problèmes. L'information de tout le projet est alors regroupée dans un même formalisme et elle est rapidement accessible grâce à un système d'interrogation. Ainsi il est plus aisé de produire un gros logiciel et d'en effectuer la maintenance. Le système ELEN (géniE logicieL et recherchE d'informatioNs) a donc pour objectif la gestion et l'interrogation des logiciels et de leurs documentations associées. Dans cet article, nous nous intéressons plus particulièrement à la fonction d'interrogation des codes sources, qui est fondée sur une extension du modèle des graphes conceptuels.

Mots Clés : Système de Recherche d'Informations, Réutilisation de composants logiciels, Graphes Conceptuels.

Abstract :

The increasing life and size of software exhibits a lot of problems dealing with software engineering activity which are : consistency between documents and software, versions management, parcelling information between the project staffs, verification and accuracy of software, retrieval of reusable codes, etc. A complete and uniform software management with their documents and the possibility to retrieve these managed informations, can decrease lots of problems. All the project information are grouped together in a single formalism and are quickly accessible with a retrieval system. By this way it is easier to produce large softwares and to maintain them. The goal of ELEN system is to manage software and to retrieve software information and their own documentation. In this paper, we focus on the design of a source code retrieval system which is based on an extension of conceptual graph model.

Keywords : Information Retrieval System, Software Reuse, Conceptual Graphs.

1 INTRODUCTION

Le génie logiciel est un domaine de l'informatique où l'on manipule de grosses quantités d'informations. La taille d'un logiciel important peut se calculer en millions de lignes de code. La gestion de cette masse d'informations se fait au moyen d'ateliers de Génie Logiciel (voir par exemple [Estublier 90]), véritables bases de données spécialisées dans la gestion des codes sources de grands logiciels. A l'aide de ces systèmes, il est possible de constituer automatiquement des programmes exécutables à partir d'une sélection de codes sources (i.e. modules) sur des attributs prédéfinis (auteur, date d'une version, langage de programmation, etc). Ces ateliers sont des plate-formes de travail auxquelles viennent s'ajouter des outils facilitant le travail des développeurs [Favre 88, Ambras 88].

Le code source d'un logiciel est indissociable des documents qui ont servi à son élaboration (documents de spécification, de conception, de réalisation, etc). Ces documents doivent également trouver leur place dans les ateliers de développement de logiciels [Jarwah 90].

La quantité importante d'informations produites (documents et logiciels) au cours du cycle de vie d'un logiciel et leur gestion dans une base commune, conduit à l'utilisation d'outils pour retrouver un document par la connaissance seule de son contenu, pour des activités liées à la conception (ex : réutilisation de code) et à la maintenance (ex: localisation de code pour des corrections).

Pour les documents textuels, les techniques éprouvées de recherche d'informations [Rijsbergen 79, Salton 89] peuvent être utilisées en tenant compte de l'aspect dynamique et de la structure particulière de ces documents.

Pour les logiciels eux-mêmes, des techniques spécialement adaptées doivent être mises en place pour tenir compte des spécificités propres à ces documents.

Dans le projet E.L.E.N. [Jarwa 89], nous cherchons d'une part à intégrer tous les documents (documents textuels et logiciels) issus du cycle de vie d'un logiciel dans un même système de gestion de l'information, et d'autre part nous proposons un système d'interrogation des logiciels par leur contenu.

Dans ce document, nous allons principalement présenter les aspects liés à l'interrogation d'un corpus de codes sources de logiciels. Pour cela nous présenterons tout d'abord le contexte dans lequel se place notre outil

d'interrogation; puis nous exposerons le principe des systèmes de recherche d'informations et quelques applications au Génie Logiciel. Nous décrirons ensuite le modèle d'indexation. Finalement la dernière partie abordera quelques aspects liés à la réalisation du prototype.

2 UN SYSTèME D'INTERROGATION POUR LE GÉNIE LOGICIEL

Dans cette partie, nous allons présenter quelques domaines du Génie Logiciel propices à l'utilisation d'un système d'interrogation du code source par son contenu sémantique.

2.1 La phase de développement

Une étape importante de la phase de développement d'un logiciel est celle de sa conception qui correspond à la description complète de l'ensemble des *objets logiciels* (procédures, méthodes, etc) qui le composent. Dans une optique de programmation modulaire, la fin de cette phase produit une décomposition du logiciel en modules avec leurs interfaces décrites en détail. Dans un contexte de programmation orienté objet, la fin de cette phase est atteinte avec la description des classes et des méthodes associées.

Un atelier de Génie Logiciel gère toutes ces entités et les rend accessibles grâce à des requêtes combinant des attributs prédéfinis (date, auteur, etc). Des outils, comme le Masterscope [Xerox 85], permettent d'exprimer des requêtes portant sur le contenu du code source. L'élaboration de la réponse à ce type de requêtes nécessite une analyse du texte source. Ce genre d'outil sert par exemple, à répercuter dans le logiciel les modifications de définition d'une procédure. Pour cela il recherche toutes les occurrences d'utilisation de la procédure modifiée et les propose en édition.

Cependant cette interrogation se limite aux aspects syntaxiques du contenu du logiciel. Pour augmenter la souplesse d'utilisation il faut utiliser des connaissances sur le contenu sémantique du code source. Il devient possible de s'affranchir de la connaissance de détails de conceptions et de réalisation.

> Lorsqu'un utilisateur veut connaître toutes les fonctions qui sont responsables de la fermeture d'une fenêtre graphique, avec le Masterscope sa requête est : "WHICH FUNCTIONS CALL SOMEHOW CloseWindow OR DelWindow"[1]. Cette formulation implique que l'utilisateur connaisse les noms exacts des fonctions. Si l'on ajoute au système de l'information dans des descripteurs attachés aux fonctions, la requête peut devenir : "WICH FUNCTIONS CLOSE WINDOW". Cette formulation implique que le système est capable de retrouver les noms de fonctions CloseWindow et DelWindow à partir des termes CLOSE et WINDOW et de générer la requête pour le Masterscope.

Le critère de réutilisation permet d'espérer réduire le coût de la réalisation du logiciel. La réutilisation comporte les étapes suivantes : après définition de ses besoins, il faut accéder à du code répertorié dans une *librairie* de codes réutilisables, sélectionner le code le plus proche de la fonctionnalité recherchée, comprendre ensuite le code sélectionné et finalement adapter ce code à ses propres besoins. Il est donc nécessaire de disposer d'un système permettant d'exprimer la fonctionnalité dont le concepteur a besoin, et capable de sélectionner l'élément de la librairie qui correspond le plus à ce besoin.

En conclusion, un outil d'interrogation par le contenu sémantique d'un logiciel est indispensable à la réutilisation de code. De plus, il peut élargir le champ d'application des outils de manipulation d'un atelier de développement de programmes.

2.2 La phase de maintenance

La maintenance est la dernière étape dans le cycle de vie d'un logiciel. Elle consiste à en assurer l'évolution après la livraison en corrigeant les erreurs, en l'adaptant à un nouvel environnement ou à de nouvelles contraintes, et en ajoutant de nouvelles fonctionnalités.

L'activité de maintenance est critique et indispensable. Elle est critique car ce sont rarement les même équipes qui conçoivent et maintiennent un logiciel et la durée de vie des logiciels peut être importante. Elle est indispensable car un logiciel comporte toujours un pourcentage constant d'erreurs, et du fait de son importante durée de vie, il peut être amené à changer d'environnement (autre version du système, autre machine, autre version du langage, autre langage..), et à l'usage peuvent apparaître de nouveaux besoins.

La facilité avec laquelle un logiciel sera maintenu, dépend des méthodes utilisées pour sa conception, et de la quantité et qualité de sa documentation. Les difficultés généralement constatées lors de la maintenance peuvent être imputées aux faits suivants : un manque de documentation sur les modifications successives, une difficulté à suivre le processus initial de conception, une mauvaise stabilité des programmes à cause des effets de bords, et finalement une mauvaise intégration de la maintenance dans le processus de création (voir [Schneidewind 87]).

Une première approche dans la recherche de solutions pour faciliter la maintenance est de la prévoir dès la conception, en proposant des méthodes de

travail et une politique globale de gestion. Il faut penser par exemple dès la conception, aux effets possibles des modifications ultérieures, à ne modifier qu'un module à la fois, à effectuer des tests après chaque modification, à journaliser les erreurs, à centraliser les déclarations, à créer un dictionnaire pour les définitions des termes nouveaux ou ayant un sens particulier dans le contexte du logiciel.

La seconde approche complémentaire est de proposer des outils et des méthodes qui prennent en compte une maintenance mal prévue ou difficile. Il faut typiquement dans ce cas des outils qui recréent une information implicitement disponible lors de la conception (par exemple des tables de références croisées). Un système de gestion centralisé de cette information est nécessaire avec une fonction primordiale : la localisation de code selon des critères fonctionnels [Selby 88]. La maintenance peut donc également bénéficier d'un outil d'interrogation par le contenu sémantique.

2.3 L'utilisation

L'utilisation de logiciels ou de systèmes d'exploitation comportant un très grand nombre de fonctionnalités est difficile pour un utilisateur novice. Les fonctionnalités d'un système d'exploitation sont accessibles à travers des *commandes* que l'on active en indiquant un nom suivi de paramètres à un interpréteur de commandes [Krakowiak 88]. L'utilisateur du système doit rechercher parmi les commandes disponibles, celles qui correspondent à la fonction qu'il veut activer. Si l'on ramène les commandes d'un système au niveau des éléments d'un logiciel, on réduit le problème à la recherche d'éléments d'un logiciel par leur contenu.

2.4 Conclusion

Les activités du Génie Logiciel, particulièrement dans les domaines de la maintenance et la réutilisation, nécessitent la manipulation de grandes quantités de documents de natures différentes (textuelles, logicielles, graphiques, etc) ainsi que la *recherche d'informations* dans cet ensemble de documents. Nous allons nous intéresser plus particulièrement à cet aspect "recherche documentaire" dont nous décrirons l'application à la recherche de composants logiciels satisfaisant certaines fonctionnalités, en vue de leur réutilisation ou de leur maintenance. La partie suivante introduit donc le domaine de la Recherche d'Informations avec des exemples d'applications à la réutilisation de code.

3 APPLICATION DES S.R.I. À LA RÉUTILISATION DE CODE

Les systèmes de recherche d'informations (S.R.I.) ont pour origine la recherche de documents dans une base de références bibliographiques appelée *corpus*. Plus récemment certains systèmes gèrent non pas la référence mais le document lui-même; de plus les types de documents se diversifient ou se spécialisent [Berrut 90, Nie 90] (images, sons, programmes, etc).

3.1 Le schéma de principe

Le schéma général de principe d'un S.R.I est donné figure 1. L'utilisateur recherche un document à partir d'informations de natures différentes : des informations ou *attributs* externes comme le titre du document, les auteurs ou la date de création, et des informations dites internes qui portent sur le contenu du document. Les requêtes permises peuvent faire référence à ces deux types d'informations.

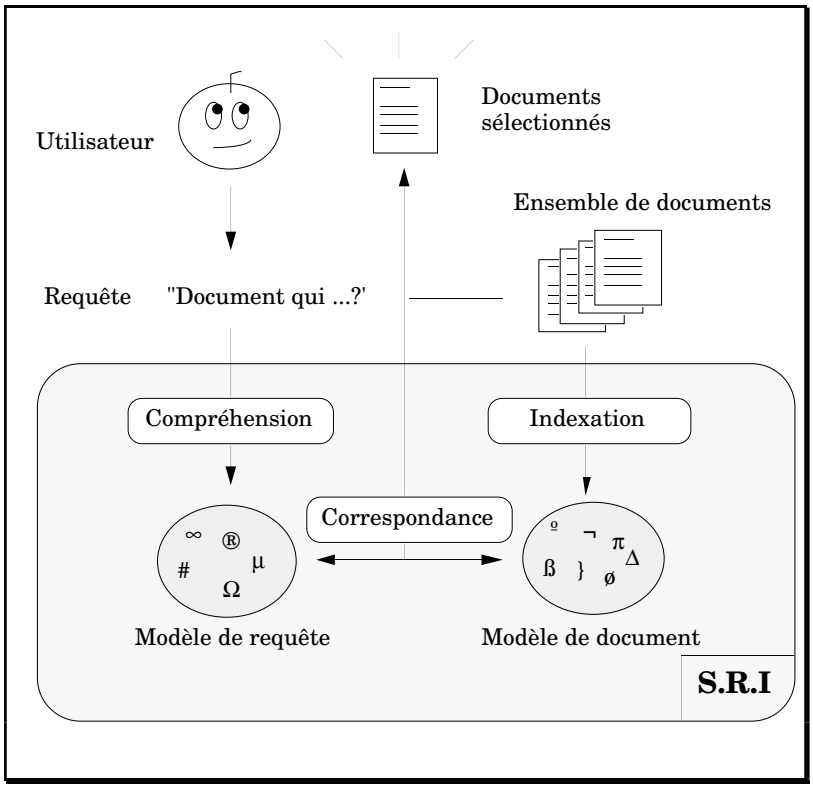


Figure 1

Le S.R.I va alors résoudre la requête en proposant à l'utilisateur l'ensemble des documents susceptibles de la satisfaire. Cette comparaison s'établie au moyen d'une *fonction de correspondance* appliquée aux attributs internes et externes. La définition de cette fonction varie d'un système à l'autre, et

l'ensemble des réponses à une requête de l'utilisateur diffère donc selon les systèmes. La plupart du temps la fonction de correspondance n'est appliquée que sur une sélection de documents qui vérifient les attributs externes.

Pour rendre cettecorrespondance possible, un *modèle de document* est utilisé, permettant d'associer à chaque document un *descripteur*. De même la requête est associée à une instance d'un *modèle de requêtes*. La correspondance a ainsi lieu au niveau de ces deux modèles. Sous certaines conditions, l'indexation des documents peut être automatique [Salton 89]. La plupart du temps, il n'y a pas de différence entre ces deux modèles.

Nous allons voir maintenant quelques exemples de techniques de recherche d'informations appliquées à la réutilisation de code.

3.2 La classification par facettes

Prieto-Diaz [Prieto-Diaz 90] propose une méthode basée sur une classification des composants logiciels. Cette classification se fonde sur la définition d'attributs ou facettes permettant de décrire un composant. Ces attributs ont une organisation hiérarchique. Indexer un composant de logiciel revient à sélectionner les facettes qui lui correspondent le mieux.

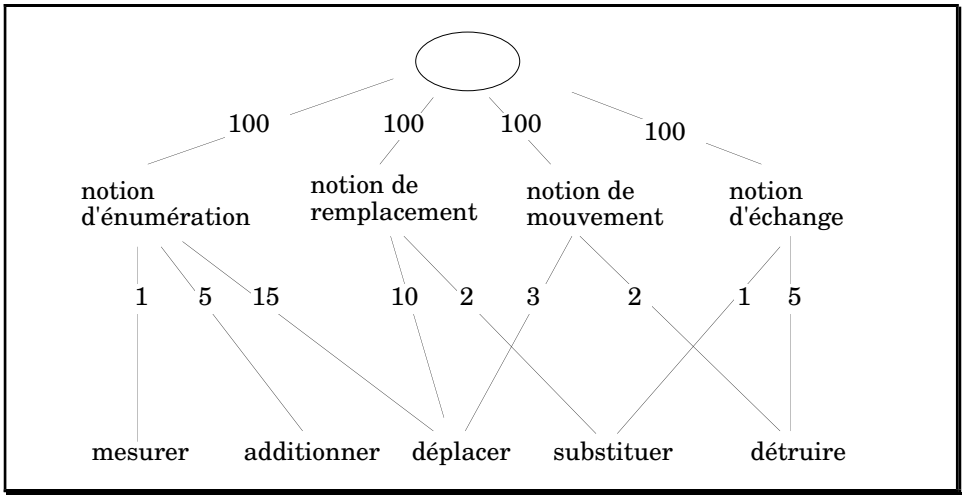


Figure 2

Les facettes utilisées en Génie Logiciel sont organisées selon la notion de fonctionnalité du composant (ce qu'il fait), selon son environnement (où il le fait) et éventuellement selon sa réalisation (comment il le fait). Un exemple de facettes est donné dans [Prieto-Diaz 87] : fonction, objet et médium. La facette "fonction" est un verbe exprimant l'action principale que réalise le programme (ex: mesurer, détruire, trier); la facette objet représente les objets manipulés par

le programme (caractères, lignes, dessins); la facette médium représente les entités supports de l'action réalisée (structures de données).

Un thésaurus est utilisé pour unifier sous un seul terme les synonymes. Un graphe de concepts (cf fig 2) permet d'exprimer des distances sémantiques entre concepts, afin d'améliorer les performances lors de l'interrogation. En effet, ce graphe est utilisé pour faire des déductions : lorsqu'une requête ne correspond à aucun composant logiciel de la base, certains termes sont remplacés par des termes sémantiquement proches. Cette distance sémantique est calculée en faisant la somme des poids des arcs du chemin entre les deux concepts. On choisit systématiquement le chemin de plus faible poids.

Par exemple, si la requête représentée par le triplet : <déplacer,fenêtre,écran> ne donne aucune réponse, le système remplacera le terme *déplacer* par *détruire* distant de 5 unités.

Le choix des facettes et les termes possibles pour chaque facette ainsi que la construction du thésaurus et du graphe de concepts est effectué par les spécialistes du domaine. La classification par facettes s'apparente à une méthode d'indexation booléenne, fondée sur un modèle intelligent de recherche d'informations. En effet le descripteur d'un composant logiciel est une conjonction de mots-clés, et un ensemble de connaissances sur le sens des termes d'indexation permet de transformer la requête par déduction.

3.3 L'indexation automatique au travers de documents

Des systèmes d'analyse automatique des textes sources de logiciels pour en extraire le contenu sémantique existent [Soloway 85]. Cependant ils ne fonctionnent que sur des codes de petite taille et dans des domaines restreints. Ces systèmes ne sont donc pas actuellement opérationnels pour indexer une vaste librairie de codes. Il est par contre possible d'utiliser les documents textuels qui accompagnent les logiciels : les techniques classiques d'indexation automatique de textes en langue naturelle peuvent être utilisées pour indexer ces documents. Grâce à la liaison qui existe entre un composant logiciel et sa documentation, on peut alors associer l'index obtenu à partir du document au composant logiciel.

Le système GURU [Maareck 89a, 89b] extrait automatiquement des *groupes conceptuels* du contenu d'un document, qui après sélection constituent les termes d'indexation du document. Cette extraction est réalisée en analysant les

relations lexicales entre les mots. Deux termes sont en relation lexicale s'ils sont en corrélation au sein d'une unité syntaxique. Des exemples de relations lexicales sont les liaisons entre un sujet et un verbe, entre un verbe transitif et son complément d'objet direct ou entre un nom et son adjectif.

3.4 Conclusion

La sémantique des objets logiciels recherchés (modules, fonctions, procédures, etc) est par définition sans ambiguité, ce qui n'est en général pas le cas des documents en langue naturelle. Il est cependant difficile de mettre à profit cette caractéristique car l'analyse automatique des textes sources des programmes pour en extraire des informations sémantiques n'est pas réalisable actuellement. La solution intermédiaire semble donc se trouver dans l'expression d'une information se plaçant entre le code source et la documentation en langue naturelle. Cette information, qui s'apparente à une *spécification,* devra être fournie par le programmeur lui même, assisté éventuellement par des outils spécifiques capables d'analyser la documentation et les codes sources des logiciels. L'architecture du fonctionnement d'un tel système est schématisée par la figure 3 ci-dessous.

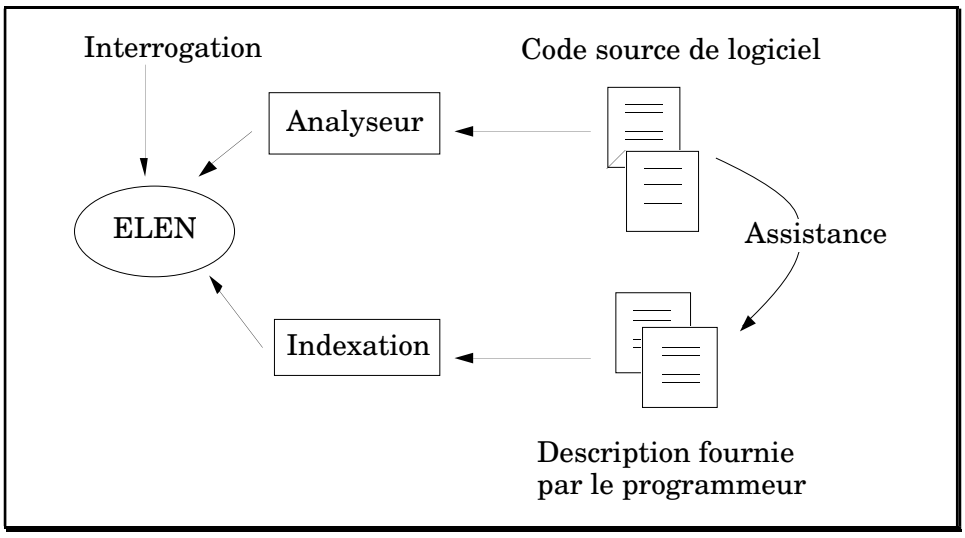


Figure 3

Notre objectif est de favoriser la précision des réponses, c'est à dire d'augmenter la proportion des documents pertinents retrouvés dans le nombre total de réponses données. Pour cela nous avons choisi un modèle basé sur l'expression du contenu sémantique. Ce modèle doit permettre d'exprimer une taxonomie de concepts, des relations sémantiques, ainsi que des schémas décrivant les formes possibles de relations entre ces concepts. C'est pourquoi

nous nous sommes orienté vers le modèle des graphes conceptuels. Ce modèle que nous présentons dans la section suivante dans ses grandes lignes, est particulièrement intéressant car il comporte des opérateurs permettant d'effectuer des déductions sur les connaissances. Il nous servira à exprimer les descripteurs et les requêtes.

4 LE MODèLE DES GRAPHES CONCEPTUELS

Ce modèle de représentation de la connaissance a été introduit par Sowa [Sowa 84]. Cette partie en rappelle les fondements.

4.1 Les concepts et les relations conceptuelles

Sowa met en oeuvre deux notions de base : les concepts et les relations conceptuelles. Ces deux notions sont les noeuds d'un graphe fini connexe et biparti[2] appelé graphe conceptuel.

Par définition un concept, représenté graphiquement par une boite, a un **label de type** noté en majuscule, que nous nommerons simplement **type** et un **référent** noté en minuscule (ex : [PERSONNE : jean]). Le type est élément d'un ensemble ayant une structure de treillis (ex : HOMME < PERSONNE). Dans un concept, le référent et le type doivent être liés par une *relation de conformité*.

Deux concepts sont *identiques* s'ils ont même type et même référent. Le référent d'un concept peut être :

- non défini : le concept est alors appelé *concept générique*; le concept [PERSONNE] désigne une personne en général;
- non identifié : le concept [PERSONNE : #] désigne une personne particulière;
- un ensemble : le concept [PERSONNE : {jean,luc}] désigne l'ensemble formé des personnes jean et luc;
- des quantifications : le concept [PERSONNE : ∀] désigne toute personne;
- etc.

Une relation conceptuelle représentée graphiquement dans un ovale, est identifiée par un type. L'ensemble des types des noeuds conceptuels est disjoint de l'ensemble des types des noeuds relations.

Par exemple 

est un graphe qui exprime que *jean mange la pomme*.

4.2 Les graphes canoniques

Pour limiter l'expression aux seuls graphes conceptuels ayant un *sens*, Sowa introduit la notion de graphes *canoniques*. Ces graphes expriment les combinaisons licites de concepts et de relations conceptuelles. L'ensemble des graphes représente tout ce qui est *exprimable* dans ce modèle; le sous-ensemble des graphes canoniques limite à tout ce qui est *possible*.

Certains graphes conceptuels sont déclarés canoniques *à priori* : ils forment la *base canonique*. Les autres graphes canoniques sont calculés à partir des opérations définies ci-après :

- la copie de graphe : toute copie de graphe canonique est un graphe canonique.
- la restriction : le type d'un concept est remplacé par un type inférieur, le référent doit alors se conformer au nouveau type. Par exemple, [PERSONNE : jean] peut devenir [HOMME : jean] si le type HOMME et le référent jean sont en relation de conformité.
- la jointure de deux graphes sur un concept commun : le nouveau graphe est formé par suppression d'un des deux concepts communs et par attachement des parties pendantes à l'autre concept. Les graphes [A] → (r) → [B] et [B] ← (r) ← [C] se joignent sur le concept [B] et donnent le graphe [A] → (r) → [B] ← (r) ← [C].
- la simplification : elle supprime les relations conceptuelles redondantes entre deux concepts. Le graphe [A] → (r) → [B] ← (r) ← [A] se simplifie en [A] → (r) → [B].

4.3 La définition des types et des relations

Il est possible de *définir* un type de concept en lui associant tout un graphe conceptuel. Cette définition permet d'exprimer un type de concept ou de relation en fonction de types plus *élémentaires*. Dans l'exemple de la figure 4, le concept OBJET-LOOPS a été défini comme une instance d'une classe Loops. Un des référents des concepts du graphe, repéré par la variable x, devient le référent du nouveau concept.

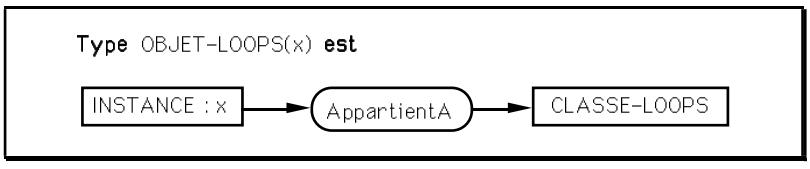


Figure 4

De même, il est possible de définir une nouvelle relation conceptuelle en lui associant tout un graphe et en désignant des concepts particuliers de ce graphe.

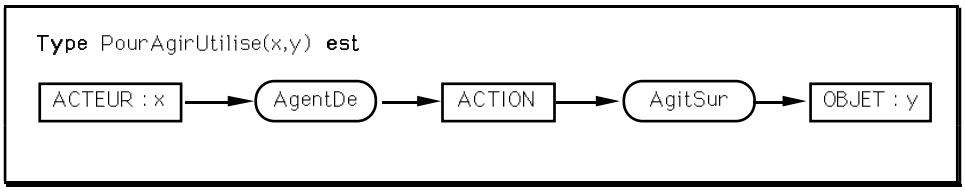


Figure 5

L'exemple de la figure 5 contient la définition d'une relation conceptuelle entre les types de concepts ACTEUR et OBJET.

4.4 La projection

Sowa a défini un ordre partiel sur les graphes conceptuels à partir des opérations données au paragraphe 4.2. La définition de cet ordre est :

Soient G et G' deux graphes conceptuels, G' ≤ G si G' résulte des opérations de copie, restriction, jointure et simplification appliquées à G.

A partir de cet ordre partiel est défini la *projection* qui permet d'extraire des sous-graphes d'un graphe conceptuel. Sa définition est la suivante :

Soient deux graphes conceptuels G et G' tels que G' ≤ G. Un graphe résultant de la projection de G sur G' est défini comme un sous graphe de G' :

- qui possède les mêmes relations conceptuelles que G,
- dont les concepts sont des restrictions de ceux de G

Le résultat d'une projection n'est pas unique. Cette opération, on le verra plus loin, est utile au mécanisme de recherche d'informations.

5 L'UTILISATION DES GRAPHES CONCEPTUELS

Nous avons indiqué dans le paragraphe 3.4 l'intérêt des taxonomies de concepts, des relations sémantiques et des schémas types de relations entre concepts. Le modèle des graphes conceptuels permet d'exprimer une taxonomie de concepts grâce à la relation d'ordre partiel entre concepts, il permet également d'exprimer des relations sémantiques et des schémas grâce à la définition de types de concepts.

5.1 L'indexation

Nous avons choisi ce modèle pour exprimer les descripteurs attachés aux objets logiciels. Un descripteur est alors un graphe conceptuel canonique. Cela permet de restreindre l'ensemble des descripteurs possibles à la fermeture de la base canonique sur les opérations de copie, de jointure, de restriction et de simplification.

L'opération d'indexation d'un objet logiciel consiste donc à construire un graphe canonique et à l'associer à cet objet.

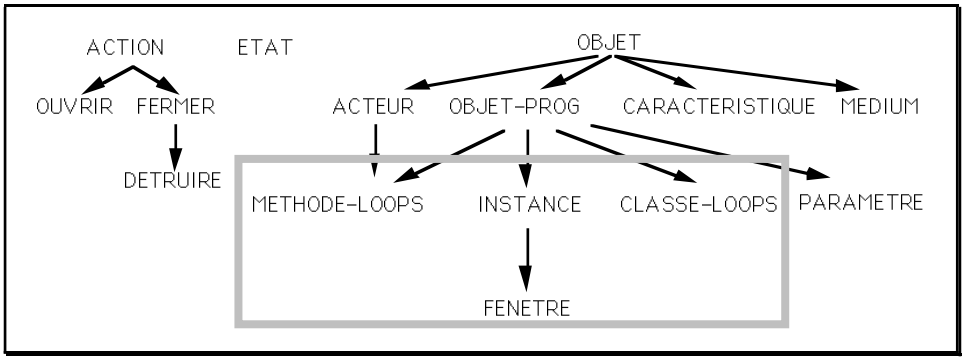


Figure 6

Nous avons étudié les commentaires des méthodes du système LOOPS et la documentation des commandes du système Unix. Nous en avons extrait une hiérarchie de types de concepts dont une partie est donnée figure 6, un ensemble de référents attachés à ces types et un ensemble de graphes canoniques de base. Cette connaissance est définie *à priori* pour un domaine de l'informatique et en collaboration avec un expert de ce domaine.

La figure 7 présente une partie du treillis des relations conceptuelles. Le cadre englobant des concepts de la figure 6 représente le domaine des concepts propres à l'environnement LOOPS.

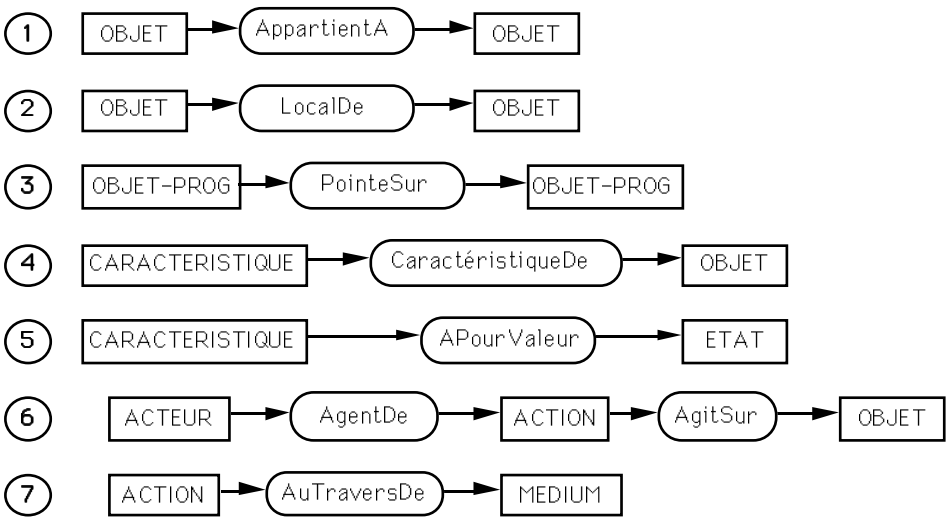
Figure 7

La liste ci-après présente quelques types de concepts :

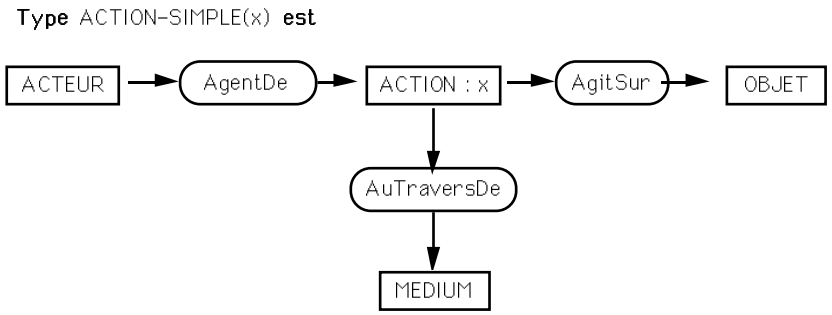
- OBJET désigne tout objet "passif" existant dans un logiciel (fenêtre, point, etc);

- ACTION exprime au travers d'un groupe verbal, une action que l'on fera porter sur un OBJET comme OUVRIR, FERMER, DETRUIRE;
- ACTEUR désigne tout objet agent d'une action (procédure, utilisateur, etc);
- MEDIUM désigne le support d'une action (structure de données, objet physique de transmission, etc);
- OBJET-PROG designe un objet de programmation; le référent particularise si nécessaire le concept en donnant le nom de l'identificateur de l'objet de programmation; dans [PARAMETRE : self] de la figure 8, PARAMETRE est un sous concept de OB-PROG et self est le nom d'un identificateur dans le langage LOOPS;
- CARACTERISTIQUE donne des renseignements sur un concept individuel de type OBJET comme son état (aspect, taille,...);
- ETAT exprime l'état d'une caractéristique d'un objet (ouvert, grand,...).

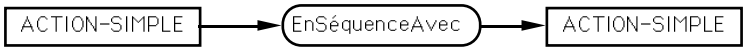
Nous donnons ci-après quelques graphes canoniques de base.



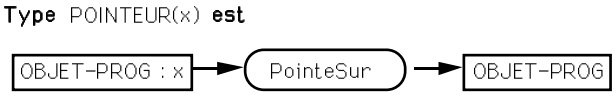
Le type de relation conceptuelle LocalDe se spécialise en sous-types ACotéDe, IntérierieurDe (voir fig.7). La jointure des deux derniers graphes donne un graphe canonique qui est utilisé pour la définition du type ACTION-SIMPLE.



Une action simple exprime les *fonctionnalités* de base que l'on peut exprimer dans ce modèle.



Ce graphe canonique permet d'exprimer la notion de séquence d'action simples.



Le type de concept POINTEUR est défini à partir du graphe canonique utilisant la relation conceptuelle PointeSur. Dans la figure 8, self peut alors devenir le référent d'un concept de type POINTEUR.

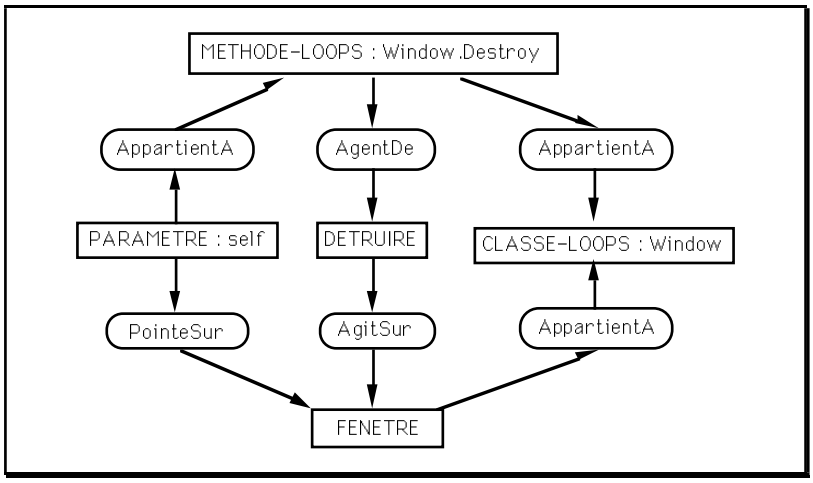


Figure 8

Cette figure propose un exemple d'indexation d'une méthode du système LOOPS. La méthode Close de la classe Window ferme une fenêtre graphique passée par le paramètre self :

(<- *self* **Close**) Méthode de la classe Window.

Comportement : ferme la fenêtre self.

Paramètre : *self* pointeur sur une instance de laclasse Window.

Ce graphe est obtenu par jointure et restriction des graphes canoniques ①, ③ et ⑥.

5.2 L'interrogation

L'interrogation a pour objectif de trouver l'objet logiciel dont le descripteur *correspond* à la requête. Une requête est un graphe conceptuel canonique dont un concept apour référent le symbole '?'. Seuls certains types de concepts comme

METHODE-LOOPS peuvent avoir ce symbole en référent : ce sont les types de concepts dont le référent possède un descripteur.

L'évaluation d'une requête passe par l'opération de projection. Pour que l'opération de projection tienne compte de la taxonomie sur les relations conceptuelles, nous étendons sa définition qui devient alors :

Soit deux graphes conceptuels G et G' tels que G' ≤ G. Un graphe résultant de la projection de de G sur G' est défini comme un sous graphe de G' :

- dont les relations conceptuelles sont des restrictions de celles de G,
- dont les concepts sont des restrictions de ceux de G.

La figure 9 montre en grisé le résultat de la projection du graphe G représentant la requête, sur le graphe G' descripteur de la méthode Close. Le concept [METHODE-LOOPS : ?] a été restreint au concept [METHODE-LOOPS : Window.Close] et la relation AgitSur a été restreinte en ChangeEtatDe. La requête G peut également se projeter de manière identique sur le graphe associé à la méthode Destroy de la figure 8 car dans le treillis des types de concepts, DETRUIRE est inférieur à FERMER.

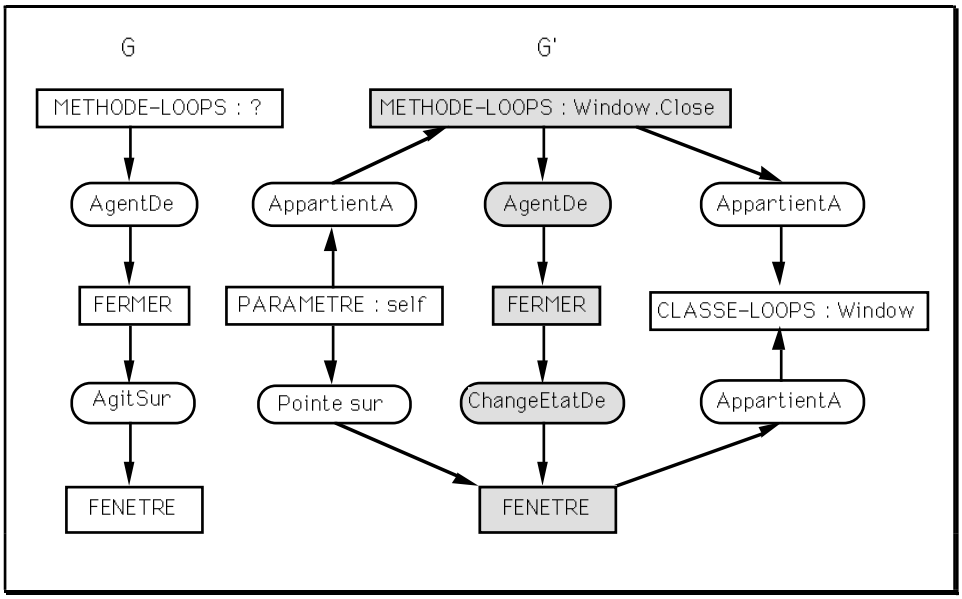


Figure 9

L'opérateur de projection ne donne pas toujours un résultat unique. Notre objectif est de rechercher le descripteur qui correspond le mieux à une requête. Il nous faut donc une fonction d'évaluation du résultat de la projection. Nous

proposons la définition d'une fonction d'évaluation ∂ ayant les propriétés suivantes :

Soient deux graphes conceptuels G et G', G' ≤ G et soit G'' un graphe résultat d'une projection de G sur G' :

- si G = G'' <=> ∂(G,G'') = 0,
- sinon ∂(G,G'') ≥ 0.

Plusieurs fonctions d'évaluation sont possibles comme par exemple, la prise en compte du nombre de noeuds de la partie pendante du résultat de la projection (soit 6 dans l'exemple) ou bien la prise en compte du nombre de noeuds concepts différents du concept de référent '?', qui ont été restreints (soit 1 dans l'exemple). Pour deux graphes identiques, le graphe résultat de la projection est le graphe initial, les graphes pendants sont donc vides et leur nombre de noeuds est nul.

Finalement il faut évaluer la distance f entre la requête et les descripteurs (G, G'). Elle peut correspondre au minimum de l'évaluation de ∂ entre G et toutes les projections possibles G''. La sélection d'un graphe conceptuel satisfaisant au mieux la requête G est alors le graphe qui minimise la fonction f(G,G').

6 EXPÉRIMENTATION

Le prototype ELEN en cours de réalisation possède un niveau interne qui implante les mécanismes des graphes conceptuels décrits précédemment. Il possède également un niveau externe qui facilite l'expression des requêtes et des descripteurs en masquant à l'utilisateur leur structure de graphe conceptuel. Ces deux niveaux se manipulent à l'aide d'une interface graphique interactive.

Le niveau externe permet d'exprimer des graphes de termes appartenant à un dictionnaire. Ce dernier associe à un terme, un ou plusieurs types de concepts. Pour constituer ce dictionnaire nous avons sélectionné parmi tous les verbes de la langue française, ceux dont la sémantique *a le plus de rapport* avec le domaine du Génie Logiciel. Il renferme environ 300 verbes répartis dans des catégories hiérarchisées associées à des sous-types de concepts de ACTION. De même une sélection de termes exprimant des objets, des caractéristiques et des états possibles, a été réuni dans ce dictionnaire.

Le niveau interne est celui des graphes conceptuels. Le système doit alors transformer un graphe de termes (externe) en graphe de concepts (interne).

6.1 Les sous concepts des actions

Nous avons classé ces verbes dans des sous concepts de ACTION et nous donnons ci-après quelques exemples :

Changer-état : porte sur un objet. Ces verbes expriment qu'une partie de l'état de l'objet a pu être modifié. Le changement d'état peut être vu comme une fonction qui prend en entrée l'objet et qui le rend modifié. Un ensemble d'états est fourni pour certains verbes.

Créer : porte sur un objet. Ces verbes expriment qu'un objet du même type que l'objet de l'action, est créé. Ce sont des sous concepts de Changer-état.

Copier : c'est un sous-concept de Créer, il y a en plus l'idée que l'objet créé est obtenu par une copie d'un autre objet.

Structurer : exprime qu'un objet est créé dans une structure. Cela implique que l'objet représentant la structure est modifié.

6.2 Réalisation

Le prototype est réalisé dans le langage orienté objet LOOPS [Xerox 88] qui est une sur-couche du langage LISP. Ce type de langage facilite le prototypage car il permet de passer outre les préoccupations de gestion des structures des données. Dans ce prototype, les concepts et les relations sont des objets de Loops.

Les types sont réalisés sous forme de classes, cela permet d'utiliser la structure hiérarchique et les mécanismes d'héritage entre les classes. Tous les graphes conceptuels sont des sous-classes de *ConceptualGraph* , elle même sous-classe de *Graph,* classe générique de tous les graphes manipulés dans l'application.

La gestion des attributs externes est confiée au Masterscope [Xerox 85]. Il réalise une analyse des textes sources Loops pour en extraire une table des références croisées des objets LISP et LOOPS. Le prototype sépare alors des requêtes, les attributs externes des attributs internes. Il forme avec les attributs externes une requête pour le Masterscope. C'est sur cet ensemble de fonctions et de méthodes sélectionnées que le prototype réalise sa recherche.

Le corpus des documents est formé par des méthodes du système LOOPS et par une sélection de commandes du système Unix.

7 CONCLUSION

Dans ce document nous avons montré l'intérêt que représente un système d'interrogation des codes sources d'un logiciel par l'intermédiaire de descripteurs, tant dans la conception que dans la maintenance. Pour réaliser cet outil, nous avons mis en oeuvre des modèles issus de la Recherche d'Informations et de l'Intelligence Artificielle. En l'occurrence, nous avons choisi et étendu le modèle des graphes conceptuels pour indexer les composants logiciels. Après avoir adapté l'opérateur de projection pour l'interrogation, il nous reste à évaluer par la pratique l'efficacité du calcul de distance entre une requête et un document.

Le modèle des graphes conceptuels a été conçu pour traiter la langue naturelle. Cela nous permet d'envisager des extensions futures visant à automatiser partiellement le processus d'indexation à partir de documents de spécification en langue naturelle restreinte.

RÉFÉRENCES

[Ambras 88] James Ambras, Vicki O'Day, *Microscope : A Knowledge-Based Programming Environment,* IEEE Software, May 1988.

[Berge 73] Claude Berge, Graphes et hypergraphes, N° 604 Col. Mathématique, Ed. Dunod université, 1973.

[Berrut 90] Catherine Berrut, *Indexing Medical Reports : The RIME Approach*, Information Processing & management, Vol 26, N° 1, p. 93-109, 1990.

[Estublier 90] Jacky Estublier, Noureddine Belkhatir, *ADELE 2 :Un outil pour la gestion des logiciel*, Rev. Génie Logiciel, N°21 Dec 90, p 4-15.

[Favre 88] Jean-Marie Favre, *Representation multi-langages des programmes pour la programmation globale.* Rapport de D.E.A Informatique. Université Joseph Fourier, Institut National Polytechnique de Grenoble. Présenté le 30 Juin 1988.

[Jarwah 89] Sahar Jarwah, Jean-Pierre Chevallet, *Spécifications d'ELEN : un système pour la gestion et l'interrogation des documents et des logiciels*. Toulouse 89, Deuxièmes journées internationales, le génie logiciel et ses applications. 4-8 décembre 1989.

[Jarwah 90] Sahar Jarwah, Marie-France Bruandet, *A Hypertext Database Model for Information Management in Software Engineering.* Proc. on Database and Expert Systems Applications, DEXA 90, Wienne Autriche, p69-75, Springer-Verlag.

[Krakowiak 88] Sacha Krakowiak, *Principles of Operating Systems,* The Massachusetts Institute of Technology Press, 1988.

[Maareck 89a] Yoëlle Maareck, Daniel M. Berry, Gail E. Kaiser. *Automatically generating software libraries without pre-encoded knowledge*. Research Report RC 14990 (#66631) 8/28/89 IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598.

[Maareck 89b] Yoëlle Maareck, Frank A. Smadja, *Full Text Indexing Based on Lexical Relations An Application : Software Libraries,* Proc ACM-SIGIR 1989, p 198-206.

[Nie 90] Jianyun Nie, Yves Chiaramella, *A Retieval Model based on an Extended Modal Logic and its Application to the RIME Experimental Approach,* Proc. ACM SIGIR 90, 5-7 Sept 90 Bruxelles, p 25-43.

[Prieto-Diaz 87] Ruben Prieto-Diaz, *Classifying Software for Reusability,* IEEE Software 1987,p 6-16.

[Prieto-Diaz 90] Ruben Prieto-Diaz, *Implementing Faceted Classification for Software Reuse,* Proc. 12th International Conference On Software Engineering, march 26-30, 1990-Nice, France, IEEE Computer Society Press, p 300-304.

[Rijsbergen 79] C.J. van Rijsbergen, *Information Retrieval*, second Edition, Butterworth 1979.

[Salton 89] Gerard Salton, *Automatic Text Processing,* Reading Addison-Wesley, 1989.

[Schneidewind 87] Norman Schneidewind, *The State of Software Maintenance* IEEE Transaction on software engineering, vol 13, No 3, march 1987.

[Selby 88] Richard W. Selby, Victor R. Basili, *Error Localization During Software Maintenance : Generating Hierachical System Descriptions from the Source Code Alone,* Proc 8th Conf. on Software Maintenance, Phoenix Oct 2'-27 1988, p 192-197.

[Soloway 85] Elliot Soloway, W. Lewis Johnson, *PROUST : Knowledge-Based Program Understanding,* IEEE Transaction on Software Engineering, Vol 11, N°3, March 1985, p 267-275.

[Sowa 84] John F. Sowa, *Conceptual Structures : Information Processing in Mind and Machine,* Addison-Wesley publishing company, 1984.

[Xerox 85] Rank Xerox Corp., *Interlisp-D Reference Manual, Vol 2 : environment,* Envos Software Development Environment, Koto Release.

[Xerox 88] Rank Xerox Corp., *Envos LOOPS, user manual,* Lyric/Medley Release.

---

[1] Cette requête retrouve toutes les fonctions qui appelent directement ou indirectement les fonctions CloseWindow ou DelWindow. Une fonction en appelle directement une autre si son nom apparait dans le corps de la fonction. Une fonction en appelle une autre indirectement s'il existe une chaîne d'appels menant à la fonction.

[2] Pour une définition de ces termes voir [Berge 73]